A Computational Algebra Approach to the Reverse Engineering of Gene Regulatory Networks

Reinhard Laubenbacher * Brandilyn Stigler

Virginia Bioinformatics Institute at Virginia Tech
1880 Pratt Drive, Building XV, Blacksburg, VA 24061, USA

Abstract

This paper proposes a new method to reverse engineer gene regulatory networks from experimental data. The modeling framework used is time-discrete deterministic dynamical systems, with a finite set of states for each of the variables. The simplest examples of such models are Boolean networks, in which variables have only two possible states. The use of a larger number of possible states allows a finer discretization of experimental data and more than one possible mode of action for the variables, depending on threshold values. Furthermore, with a suitable choice of state set, one can employ powerful tools from computational algebra, that underlie the reverse-engineering algorithm, avoiding costly enumeration strategies. To perform well, the algorithm requires wildtype together with perturbation time courses. This makes it suitable for small to meso-scale networks rather than networks on a genome-wide scale. The complexity of the algorithm is quadratic in the number of variables and cubic in the number of time points. The algorithm is validated on a recently published Boolean network model of segment polarity development in *Drosophila melanogaster*.

Key words: Reverse engineering, Gene regulatory networks, Discrete modeling, Computational algebra

* Corresponding author. Tel.: 540-231-7506; Fax.: 540-231-2606
Email addresses: reinhard@vbi.vt.edu (Reinhard Laubenbacher), bstigler@vbi.vt.edu (Brandilyn Stigler).

1 Introduction

Since the advent of DNA microarray technology several techniques from mathematics, statistics, engineering, and computer science have been adapted or newly developed for the purpose of using microarray and other data to reverse engineer the structure of gene regulatory networks. An important goal of systems biology is to discover and model the causal relationships between the components of such networks and the mechanisms that govern their dynamics (Kitano, 2002). Existing reverse-engineering methods can be broadly categorized as continuous vs. discrete and deterministic vs. stochastic. Some methods aim to discover only the network topology, that is, which genes regulate which others, with a directed graph or "wiring diagram" as output, possibly signed to indicate activation or inhibition. The goal of other methods is to describe the dynamics of the network. We first describe some of these methods in order to place the one proposed in this paper into context.

Several methods have been proposed that reconstruct only the wiring diagram of the network. Bayesian network methods, first proposed in (Friedman *et al.*, 2000), and further developed in (Hartemink *et al.*, 2001), can be applied to single time points. This approach was applied in (Hartemink *et al.*, 2001) to data from 52 *Saccharomyces cerevisiae* Affymetrix chips, in order to analyze and score models of the gene regulatory network responsible for the control of genes necessary for galactose metabolism. The network considered involved 7 nodes. In (Filkov *et al.*, 2002) another statistical method was proposed that compares expression profiles from a time series of measurements. As an application the authors analyzed the data sets on *S. cerevisiae* in (Cho *et al.*, 1998) and (Spellman *et al.*, 1998), which consist of four microarray time series.

A method adapted from metabolic control analysis was proposed in (de la Fuente *et al.*, 2002). It requires input data based on very small perturbations of the rate of expression of each gene. The method was applied to simulated data using a model, published in (Mendoza *et al.*, 1999), of the gene network that controls flower morphogenesis in *Arabidopsis thaliana*, involving 10 genes. Using 10% perturbations, the authors were able to completely reconstruct the network of regulatory relations. While this survey of methods to reverse engineer gene networks in the form of a wiring diagram is far from complete, it provides a flavor of the variety of approaches, the scale at which the methods are being applied, and the challenges that result from a lack of appropriate experimental data.

The most common approach to the modeling of dynamics is to view a gene regulatory network as a biochemical network of gene products, typically mRNA and proteins, and to describe their rates of change through a system of ordinary differential equations. Thus, the modeling framework is that of a time-

continuous, deterministic dynamical system. We describe in detail the method in (Yeung *et al.*, 2002), as it shares many features with our method, even though it uses a different modeling framework. According to the authors the method is intended to generate a "first draft of the topology of the entire network, on which further, more local, analysis can be based." The authors make two assumptions.

Firstly, the system is assumed to be operating near a steady state, so that the dynamics can be approximated by a linear system of ordinary differential equations:

$$\frac{dx_i}{dt} = -\lambda_i x_i(t) + \sum_{j=1}^{N} w_{ij}(t) + b_i(t) + \xi_i(t),$$

for $i = 1, \ldots, N$. Here, $x_1, \ldots, x_N$ are mRNA concentrations, the λ_i are the self-degradation rates, the b_i are the external stimuli, and the ξ_i represent noise. The (unknown) w_{ij} describe the type and strength of the influence of the jth gene on the ith gene. They assemble to a square matrix W of real numbers. The output of the reverse-engineering algorithm is this matrix W. The input is a series of data points obtained by applying the stimulus $(b_1, \ldots, b_N)^T$ and measuring the concentrations $x_1, \ldots, x_N$ M times. Assembling these measurements into a matrix X, neglecting noise, and absorbing self-degradation into the coupling constants w_{ij}, we obtain a matrix equation

$$\frac{d}{dt}X = WX + B.$$

Here, X is an $(N \times M)$-matrix, W an $(N \times N)$-matrix, and B an $(N \times M)$-matrix. Using singular value decomposition (SVD) one obtains

$$X^T = UWV^T,$$

where U and V are orthogonal to each other. The first step is to obtain a particular solution W_0 to the reverse-engineering problem. One then obtains all possible solutions to the problem as

$$W = W_0 + CV^T,$$

where C ranges over the space of all square $(N \times N)$-matrices whose entries are equal to 0 for a certain range of j and arbitrary otherwise. Equivalently, CV^T ranges over all matrices that vanish on the given time points.

The second assumption made in the paper is that gene regulatory networks are sparse. This provides a selection criterion on which to base a particular choice of C, and hence of W. That is, the method selects the sparsest connection matrix W. This is accomplished through a particular choice of norm, and robust regression. The algorithm was validated by way of simulated data from three networks.

The other end of the model spectrum takes the view of a gene regulatory network as a logical switching network. First proposed in (Kauffman, 1969), Boolean network models have the advantage of being more intuitive than ODE models and might be considered as a coarse-grained approximation to the "real" network. They differ from ODE models in that time is taken as discrete and gene expression is discretized into only two quantitative states, as either present or absent. There is increasing evidence that certain types of gene regulatory networks have key features that can be represented well through discrete models or hybrid models; see, e.g., (Filkov and Istrail, 2002). Several algorithms for reverse engineering of Boolean network models for gene expression have been proposed. The algorithm REVEAL (Liang *et al.*, 1998) uses as a criterion for model selection the concept of so-called mutual information. For a given experimental data set (assumed to be already discretized into a binary data set), that is, a given set of state transitions, the algorithm finds a Boolean function for each node of the network that "optimally" determines the output from the input by using as few variables as possible. In essence, the algorithm finds the sparsest possible Boolean network that is consistent with the data. The search is done by enumeration. In order to make this process feasible, the number of inputs to the functions is restricted to at most three. The algorithm has been tested by the authors on simulated data sets and was found to perform very efficiently. Another algorithm for Boolean network identification was given in (Akutsu *et al.*, 1999). It is in essence also an enumeration algorithm, applied to networks in which the in-degree of each node is at most 2.

One of the disadvantages of the Boolean network modeling framework is the need to discretize real-valued expression data into an ON/OFF scheme, which loses a large amount of information. In response to this deficiency, multi-state discrete models and hybrid models have been developed. The most complex one (Thomas, 1991; Thieffry *et al.*, 1995; Thieffry and Thomas, 1998) uses multiple states for the genes in the network corresponding to certain thresholds of gene expression that make multiple gene actions possible. The authors are most interested in the modeling and function of feedback loops. The model includes a mixture of multi-valued logical and real-valued variables, as well as the possibility of asynchronous updating of the variables. While this modeling framework is capable of better capturing the many characteristics of gene regulatory networks than Boolean networks, it also introduces substantially more computational complications from a reverse-engineering point of view, especially the ability of asynchronous update, which adds orders of magnitude to the combinatorial explosion of possibilities. Multiple discrete expression levels were also used in the reverse-engineering method in (Repsilber *et al.*, 2002), which uses genetic algorithms to explore the parameter space of multistage discrete genetic network models.

In (Brazma and Schlitt, 2003) a hybrid modeling framework was introduced

that tries to capture discrete as well as continuous aspects of gene regulation. The authors' finite state linear model has a Boolean network type control component, as well as linear functions that represent an environment of substances that change their concentrations continuously. According to the authors, this framework can be generalized to more than two states for the logical variables. The reverse-engineering algorithm described in (Brazma and Schlitt, 2003) is based essentially on enumeration of all possible functions that fit a given data set.

Finally, it is worth mentioning a Boolean network approach that incorporates stochastic features of gene regulation. Probabilistic Boolean networks have been introduced in (Shmulevich *et al.*, 2002). While this survey of existing reverse-engineering methods is by no means comprehensive, it provides a context for the new method proposed in this paper. For more thorough reviews the reader can consult, e.g., (Bower and Bolouri, 2001) or (de Jong, 2002).

While continuous and discrete modeling approaches seem to be far apart, it is useful to keep in mind that most ODE models cannot be solved analytically and that numerical solutions of such systems involve the approximation of the time-continuous system by a time-discrete one. Furthermore, when validating an ODE model using microarray data it is often necessary to utilize thresholds for continuous concentrations. The connection between an ODE system and an associated discrete system that captures information about the continuous dynamics has been formalized in (Lewis and Glass, 1991). So, in the end, the two modeling paradigms might not be as far apart as it appears.

The modeling framework we describe in the next section is of the discrete multi-state variety. We propose here an approach that describes a regulatory network as a time-discrete multi-state dynamical system, synchronously updated. We further make the additional assumption that the set of states of the nodes in the network can be endowed with the algebraic structure of a finite number system, which allows us to use techniques and algorithms from computational algebra. As mentioned earlier, it is difficult for any reverse-engineering method to be validated using a real system as a test case, firstly because a suitable data set might not be available and, secondly, model predictions might be hard to verify without extra experiments. For this reason, almost all methods discussed above have used simulated data for validation. We have chosen to use a recent Boolean network model for segment polarity development in *Drosophila melanogaster* (Albert and Othmer, 2003), consisting of sixteen nodes per cell, representing genes and gene products. The model is sufficiently complex to be of interest, but small enough so we can compare in detail the agreements and disagreements of our reverse-engineered network with the real one, thereby illustrating the performance and limitations of our method.

2 Reverse Engineering of Polynomial Systems over Finite Fields

As mentioned above, we adopt the modeling framework of time-discrete multi-state dynamical systems. Let X be the set of possible states of the nodes of the network, and assume that X is a finite (but possibly large) set. One should think of X as a set of discretized expression levels, for instance. (We will address the issue of data discretization in a later section.) Let

$$f : X^n \longrightarrow X^n$$

be a discrete dynamical system over X of dimension n. Then f can be described in terms of its coordinate functions $f_i : X^n \longrightarrow X$, for $i = 1, \ldots, n$. That is, if $\mathbf{x} = (x_1, \ldots, x_n) \in X^n$ is a state, then $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$. We refer to such systems as *finite dynamical systems*.

The reverse-engineering problem we are focusing on is one of model selection and can be stated as follows.

Given one or more time series of state transitions, generated by a biological system with n varying quantities, choose a finite dynamical system $f : X^n \longrightarrow X^n$, which fits the data and "best describes" the biological system. To be precise, we presume that we are given a set of state transitions of the network, in the form of one or more time series. That is, we are given sequences of states

$$\begin{aligned} \mathbf{s}_1 &= (s_{11}, s_{21}, \ldots, s_{n1}), \ \ldots, \mathbf{s}_m = (s_{1m}, \ldots, s_{nm}) \\ \mathbf{t}_1 &= (t_{11}, t_{21}, \ldots, t_{n1}), \ \ldots, \mathbf{t}_r = (t_{1r}, \ldots, t_{nr}) \\ &\ldots \end{aligned}$$

These satisfy the property that, if the unknown transition function of the network is f, then

$$\begin{aligned} f(\mathbf{s}_i) &= \mathbf{s}_{i+1}, \quad i = 1, \ldots, m-1 \\ f(\mathbf{t}_j) &= \mathbf{t}_{j+1}, \quad j = 1, \ldots, r-1 \\ &\ldots \end{aligned}$$

Typically, there will be more than one possible choice. In fact, unless **all** state transitions of the system are specified, there will be more than one network that fits the given data set. Since this much information is hardly ever available in practice, any reverse-engineering method has to choose from a large set of possible network models. In the absence of a good understanding of the properties and characteristics of gene regulatory networks, one is limited to some type of Occam's razor principle for model selection. Before describing the principle we use, we first need to describe our computational framework.

We now make the further assumption that our state set X is chosen so that it can be given the structure of a *finite field* (Lidl and Niederreiter, 1997), that is, a finite number system. This is possible whenever the number of elements in X is a power of a prime number. This can be easily accomplished by an appropriate choice of data discretization. One possible approach is to choose a prime number p of possible states, in which case the number system can be taken to be $\mathbb{Z}/p$, the integers modulo p. An important consequence of this assumption is the well-known fact (see, e.g., (Lidl and Niederreiter, 1997), p. 369) that each of the coordinate functions of f can be expressed as a polynomial function in n variables, with coefficients in X, and so that the degree of each variable is at most equal to the number of elements in X.

Example. Boolean functions can be represented as polynomial functions with coefficients in $\mathbb{Z}/2 = 0, 1$. Indeed, if x and y are Boolean variables, then

$$x \wedge y = x \cdot y, x \vee y = x + y + x \cdot y, \neg x = x + 1.$$

Note that addition does not correspond to the logical OR function, but rather the exclusive OR function, XOR. Since every Boolean function can be written in terms of these three Boolean operations, we see that every Boolean function can be represented as a polynomial function on the field with two elements. Conversely, in light of the above fact, we see that any function $(\mathbb{Z}/2)^n \longrightarrow (\mathbb{Z}/2)^n$ can be realized as a Boolean network.

So assume now that our state set is a finite field, denoted k. As observed above, the model f we are searching for is determined by its coordinate functions $f_i : k^n \longrightarrow k$. We can reverse engineer each coordinate function independently and thus reconstruct the network one node at a time. Our algorithm is very similar in outline to that in (Yeung *et al.*, 2002). We first compute the space of all networks that are consistent with the given time series data. We then choose a particular network $f = (f_1, \ldots, f_n)$ that satisfies the following property:

Criterion for model selection. For each i, f_i is minimal in the sense that there is no non-zero polynomial $g \in k[x_1, \ldots, x_n]$ such that $f = h + g$ and g is identically equal to zero on the given time points.

That is, we exclude terms in the polynomials f_i that vanish identically on the data. No reverse-engineering method is able to detect such terms without prior information, even though they may exist in the real network but be nonfunctional under the particular experimental conditions. Note that this criterion is different from that used in (Yeung *et al.*, 2002) and also by REVEAL. In both of those algorithms the search is for the sparsest network, that is, a network such that each node takes inputs from as few variables as possible.

In terms of the coordinate functions, the basic mathematical reverse-engineering

problem is then as follows, formulated for one time series. The statement of the problem for several time series is similar.

Problem. Suppose we are given a collection of states $\mathbf{s}_1 = (s_{11}, \ldots, s_{n1}), \ldots, \mathbf{s}_m = (s_{1m}, \ldots, s_{nm})$, and a choice of coordinate $i \in \{1, \ldots, n\}$.

(1) Find all polynomial functions $f_i \in k[x_1, \ldots, x_n]$ such that

$$f_i(\mathbf{s}_j) = s_{i,j+1}.$$

That is, find all polynomials that map $\mathbf{s}_j$ to the ith coordinate of the next time point $\mathbf{s}_{j+1}$.

(2) From that set of functions choose one that does not contain any terms that are identically equal to zero at all time points (which is unique; see Appendix A).

The key advantage of the polynomial modeling framework over a finite field is that there is a well-developed algorithmic theory that provides mathematical tools for the solution of this problem which have polynomial time complexity in the input. Thus, we can overcome one disadvantage that discrete models have compared to ODE models, for which there is a well developed mathematical theory. This feature has been an important reason for the use of polynomial systems over finite fields in engineering, in particular control theory (see, e.g., (Marchand and Le Borgne, 1998)). We now describe the algorithm.

We fix one coordinate/node in the network and reverse engineer its transition function. To simplify notation, we assume that we have given a time series $\mathbf{s}_1, \ldots, \mathbf{s}_m$, as above, together with elements $a_1, \ldots, a_m \in k$, and we are looking for all polynomial functions $f \in k[x_1, \ldots, x_n]$ such that $f(\mathbf{s}_j) = a_j$ for all $j = 1, \ldots, m$. First we compute a particular polynomial f_0 that fits the data. There are several methods to do this, Lagrange interpolation being one of them. We use the following formula, based on the so-called Chinese Remainder Theorem (see, e.g., (Lang, 1971), p. 63):

$$f_0(\mathbf{x}) = \sum_{j=1}^{m} a_j r_j(\mathbf{x}),$$

where the polynomials r_j are defined as follows. Let $1 \leq i \neq j < m$. If $s_i \neq s_j$, then find the first coordinate l in which they differ. Define

$$b_{ij}(\mathbf{x}) = (s_{j,l} - s_{i,l})^{p-2} (x_l - s_{i,l})$$

for every $i \neq j$. Then, for $i \neq j$,

$$r_j(\mathbf{x}) = \Pi_{i=1}^{m-1} b_{ij}(\mathbf{x}).$$

If $s_i = s_j$, then we have reached a limit cycle. We can restrict the time series to the states $s_1, \ldots, s_{j-1}$. We note that $r_k(\mathbf{s}_k) = 1$ and $r_k(\mathbf{x}) = 0$ otherwise. It is

straightforward to check that this polynomial function does indeed interpolate the given time series.

Now consider two polynomials $f, g \in k[x_1, \ldots, x_n]$ such that

$$f(\mathbf{s}_j) = a_j = g(\mathbf{s}_j).$$

Then $(f - g)(\mathbf{s}_j) = 0$ for all j. That is, any two such functions differ by a function that is identically equal to 0 on the given time series. So, in order to find all functions that fit the data, we need to find all functions that vanish on the given time points. Note that the set of all such functions is closed under addition and multiplication by any polynomial function in $k[x_1, \ldots, x_n]$, and so forms a so-called *ideal* I in the set of all polynomials. In Appendix A we describe the details of an algorithm to compute generators for I (similar to a basis for a vector space). The algorithm computes polynomials $g_1, \ldots, g_r \in I$ such that any $f \in I$ can be expressed as a linear combination

$$f = \sum_{i=1}^{r} h_i g_i,$$

for some polynomials $h_i \in k[x_1, \ldots, x_n]$. The next step is to reduce the polynomial f_0 found in the previous step modulo the ideal I, that is, write f_0 as

$$f_0 = f + g,$$

with $g \in I$. Furthermore, f is minimal in the sense that it cannot be further decomposed into $f' + h$ with $h \in I$. In other words, g represents the part of f_0 that lies in I, and that therefore is identically equal to 0 on the given time series. Then we obtain all possible functions that interpolate the time series in the form $f + g$, where g runs through all elements of I. We summarize the different steps.

Reverse-engineering algorithm. (For one node of the network.)

Input: A time series $\mathbf{s}_1, \ldots, \mathbf{s}_m \in k^n$ of network states, together with expression levels $a_1, \ldots, a_m \in k$.

Output: A polynomial function $f \in k[x_1, \ldots, x_n]$ such that $f(\mathbf{s}_j) = a_j$ and such that f does not contain component polynomials that vanish on the time series.

(1) Compute a particular solution f_0 from the formula.
(2) Compute the ideal I of all functions that vanish on the data.
(3) Compute the reduction f of f_0 with respect to I.

Some comments on the features of the algorithm and its complexity are in order. Step (1) is straightforward. Step (2) is where the bulk of the computation

takes place. The first observation is that the ideal I can be computed as the intersection of the ideals

$$I_j = \langle x_1 - s_{j1}, x_2 - s_{j2}, \ldots, x_n - s_{jn} \rangle,$$

for $j = 1, \ldots, m$. Intersections of ideals can be computed algorithmically. See, e.g., (Cox *et al.*, 1997), p. 185, for details. The computation relies on the computation of a so-called Gröbner basis for several intermediate ideals. It is known that the worst-case complexity of Gröbner basis calculations is doubly exponential. In the case of finite fields, however, the calculations needed can be reduced essentially to linear algebra. Our algorithm is very similar to one used in algebraic statistics, for the selection of models for factorial design problems (Robbiano, 1998). The only difference is that our algorithm computes f_0 and I separately, whereas the algorithm in (Robbiano, 1998) combines the two. (Since I is the same for all nodes, we only need to compute it once if done separately.) It is shown there (Theorem 3.1) that the algorithm is quadratic in the number of variables and cubic in the number of time points. This algorithm is implemented in the computer algebra system CoCoA (`cocoa.dima.unige.it`). For the results in this paper we have used the computer algebra system Macaulay2 (`www.math.uiuc.edu/Macaulay2`), which uses the standard algorithms for computation of Gröbner bases. We emphasize that no portion of this algorithm is based on enumeration.

As explained in Appendix A, Gröbner basis calculations require the choice of an ordering of the terms in the polynomials, in particular an ordering of the variables. This is necessary in order to carry out long division of multivariate polynomials. The end result of the calculations depends on the term ordering chosen, in the sense that "cheaper" variables, that is, those that are smallest in the ordering, will be used preferentially in both the interpolation as well as the reduction calculations. With prior information about the existence of certain regulatory relationships this feature can be used to incorporate this prior knowledge into the algorithm. (Likewise, using the Elimination Theorem (see Appendix A), one can exclude certain variables from appearing.) In general, however, this dependence makes the interpretation of the precise form of the functions difficult at times. In those cases we resort to building a consensus model using a collection of variable orderings that makes each variable "equally costly on average." We can then extract from these different models common regulatory links and common nonlinear terms. In the next section we illustrate this strategy with a detailed discussion of an example.

3 An Application

In this section, we present an application to the well-characterized network of segment polarity genes in the fruit fly *D. melanogaster*. In (Albert and Othmer, 2003) and (von Dassow *et al.*, 2000) two models were suggested for the regulatory interactions in a network of five segment polarity genes and their products. In (von Dassow *et al.*, 2000) the authors proposed a continuous-state model using ordinary differential equations. In contrast, the Albert-Othmer model is discrete, in that the functions governing state transitions are Boolean functions. Both models were built from inferences given gene and protein expression data. Figure 1 depicts the graph of connections in the Boolean model in (Albert and Othmer, 2003). In the graph, nodes represent mRNAs and proteins. An edge between nodes indicates that the node at the head is involved in the regulation of the tail node. For example, an edge A→B between protein nodes A and B implies that A regulates the transcription and translation of B, whereas an edge A→b from protein A to mRNA b implies that A regulates the transcription of gene b. Note that edges denote the existence of regulation, not the type, whether activation or inhibition. In Table 1 which follows are some of the Boolean functions that accompany the model in Figure 1. Superscripts denote time and subscripts location relative to the current cell.

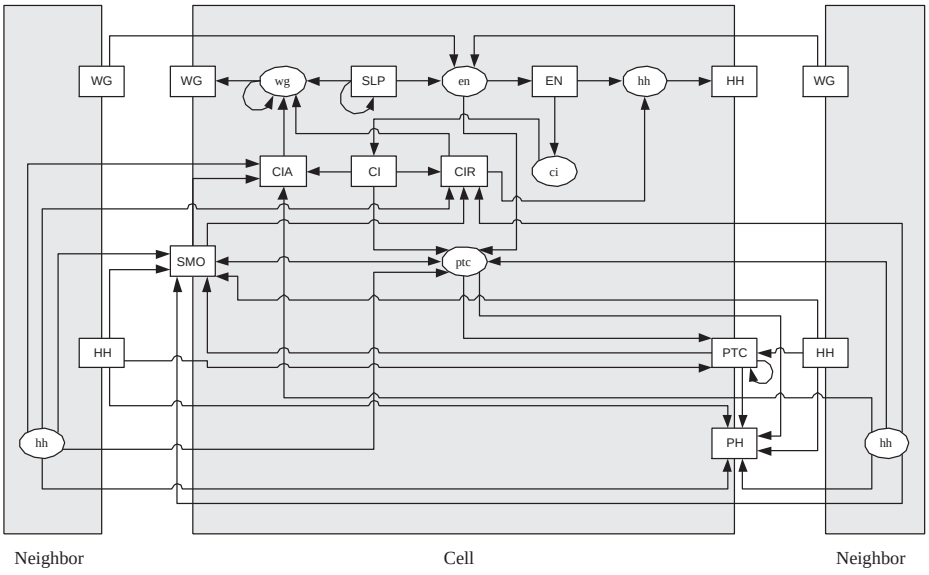

Fig. 1. The graph of interactions in the Boolean network developed in (Albert and Othmer, 2003). Ovals = mRNAs, rectangles = proteins. SLP denotes 2 forkhead domain transcription factors encoded by the gene *sloppy paired* and is believed to activate the segment polarity genes depicted in the model (Cadigan *et al.*, 1994). PH is the protein complex formed by the binding of HH to PTC (Ingham and McMahon, 2001). The protein SMO is encoded by the gene *smoothened*. Because its transcription is not regulated by any molecular species in the model, *smoothened* is not represented.

$$
\begin{aligned}
f_6 &= hh_i^{t+1} = EN_i^t \wedge \neg CIR_i^t \\
f_7 &= HH_i^{t+1} = hh_i^t \\
f_8 &= ptc_i^{t+1} = CIA_i^{t+1} \wedge \neg EN_i^{t+1} \wedge \neg CIR_i^{t+1} \\
f_9 &= PTC_i^{t+1} = ptc_i^t \vee \left(PTC_i^t \wedge \neg HH_{i-1}^t \wedge \neg HH_{i+1}^t\right)
\end{aligned}
$$

Table 1
Sample Boolean functions for the network in Figure 1.

The genes that are being modeled are *wingless* (*wg*), *engrailed* (*en*), *hedgehog* (*hh*), *patched* (*ptc*), and *cubitus interruptus* (*ci*). Our goal here is to reverse engineer the Boolean network $\mathcal{N}$ in Figure 1 from time series generated by it, including null mutant time series for each of the five genes in the network.

The network consists of a ring of 12 interconnected cells, which are grouped into 3 parasegment primordia and the genes are expressed every fourth cell. Since each cell of the ring has the same network of segment polarity genes, we focus our work on the reconstruction of the network in one cell, taking into account intercellular connections. For more details, see (Albert and Othmer, 2003; von Dassow *et al.*, 2000). Note that for our purposes it is irrelevant whether the model is indeed correct.

The starting point for our reverse-engineering algorithm is a collection of time series of discrete expression data. We used known configurations of states (Cadigan *et al.*, 1994; Hooper and Scott, 1992) for the 5 genes as initializations and generated times series for the wildtypes using the Boolean functions. All of the initializations terminate in steady states when evaluated by the Boolean functions in Table B.1 in Appendix B. To account for intercellular dependencies we included 6 extra nodes into our model. Table 2 contains the translations of the Boolean functions in Table 1 into polynomial functions in the variables $x_1, \ldots, x_{21}$, with coefficients in $\mathbb{Z}/2$. (See Table B.2 in Appendix B for all other polynomials.)

$$
\begin{aligned}
f_6 &= x_5\,(x_{15}+1) \\
f_7 &= x_6 \\
f_8 &= x_{13}\,(X + x_{21} + Xx_{21})\,(x_4+1)\,(x_{13}\,(x_{11}+1)\,(x_{20}+1)\,(x_{21}+1)+1) \\
&\quad X := (x_{11} + x_{20} + x_{11}x_{20}) \\
f_9 &= x_8 + x_9\,(x_{18}+1)\,(x_{19}+1) + x_8x_9\,(x_{18}+1)\,(x_{19}+1)
\end{aligned}
$$

Table 2
Polynomial representations of the sample Boolean functions in Table 1. See Appendix B for the legend of variable names. To account for simultaneous updating we substituted simultaneously updated terms with their function expressions (e.g. in f_8 we replaced EN with *en*).

The size of the state space of the Boolean network in each cell is 2^{21}, involving

multiple components. Any single trajectory in that space vastly underdetermines the network. If we use no known biological information other than the expression data of the wildtype to construct a discrete polynomial model, our method predicts the network to have 20 links, of which 14 are in the network ($\mathcal{N}$ has 44 links). (A description of the method with which we construct the consensus model is given below.) The performance of the algorithm dramatically improves, however, if we incorporate knock-out time series for the five genes, which we demonstrate below.

To simulate an experiment in which node x_i representing a gene is knocked out, we set its corresponding update function f_i in Table B.2 to 0 (and kept all other functions the same). We also set the corresponding functions in neighboring cells equal to 0. We then generated a new time series by iteration of the given initializations. With these data, we constructed the model in Figure 2.

Recall that we first construct all polynomial models (sets of polynomial functions) which fit a discrete time series. Then we select the one which is minimal with respect to summands in each of the functions which evaluate to 0 on all time points. The algorithm relies on the choice of a total ordering for all possible monomials in the given variables, in particular, a total ordering of the variables themselves. The effect of such a variable ordering is that the "cheaper" variables, those that are smaller in this ordering, are used preferentially in the algorithm. In order to counteract this dependency, we use a consensus model extracted from four different choices of variable ordering: the ordering $x_1 < \cdots < x_{21}$, the reverse ordering $x_1 > \cdots > x_{21}$, and 2 orderings which make "interior" variables greatest and least. Using each ordering we ran the algorithm and intersected them to obtain a consensus model (see Figure 2).

The consensus model we construct, which incorporates data from the wildtype and knock-out mutants, results in prediction of 37 of the 44 links in the Boolean network. We correctly identified the species which regulate the transcription of genes *wg* and *ci* and the synthesis of proteins SLP, WG, EN, HH, PTC, and CI. We summarize the model of interactions and compare it to the model in Figure 1. In this model we assume that transcription and translation occur in 1 consecutive time step. We will discuss reconstruction of certain dynamic properties at the end of the section.

In determining which species affect the transcription of the gene *hh*, we found a function that involves fewer terms, than the polynomial representation of its counterpart in the Boolean model. Specifically, the function is in terms of the gene product EN. The function satisfies all time series, including knock-out data, generated by its associated Boolean function. The discrepancy lies in the existence of a term in the Boolean function which does not contribute to the updating of *hh*, that is, a term that is constant equal to 0 on all input

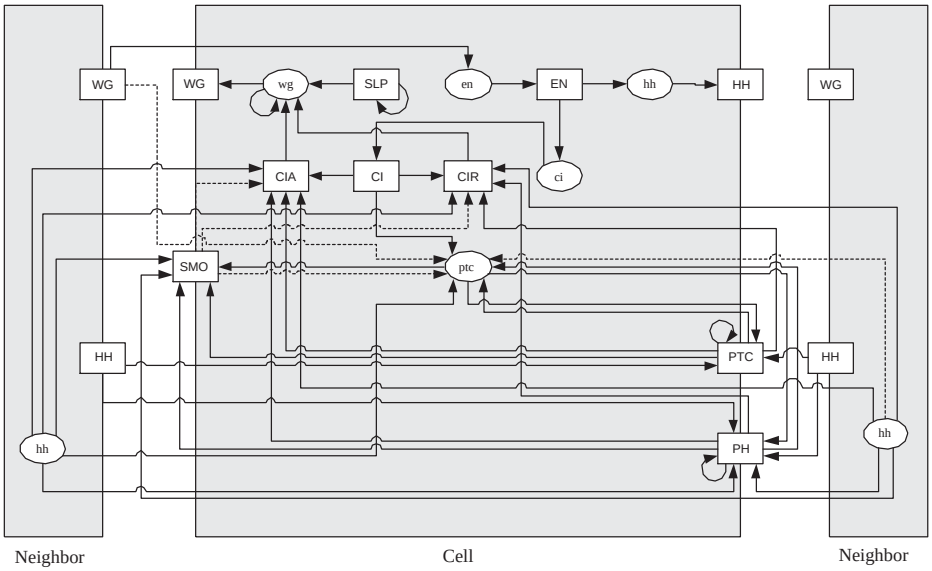

Fig. 2. The graph of the consensus model, constructed from the wildtype and knock-out time series. The graph is the intersection of 4 polynomial models, one for each variable ordering used. Dashed lines are links that appear in 3 of the 4 models.

time points. However, links whose effects are not reflected in the given data are not detectable by any reverse-engineering method unless prior information about the link is given. Similarly, the Boolean function for *en* also contains such terms, which accounts for differences in regulation dependencies. We infer that transcription of the gene *en* is regulated by Wingless protein in an anterior cell.

In this model, transcription of the *patched* gene to synthesize *ptc* mRNA in a given cell is regulated by the following molecular species: the proteins PTC, PH, SMO, CI and WG in the cell anterior, as well as *hh* from neighboring cells. We were not able to conclude that *en* regulates the transcription of the gene. It is worth observing that this regulatory link is not supported in the von Dassow model. The Albert-Othmer model predicts that transcription of the *ptc* gene is regulated only by the whole CI protein and its repressor form CIR. In fact, the authors explain that the cleavage of CI creates the repressor fragment CIR, whose synthesis is repressed by *hh* product in neighboring cells. This effect may be too indirect to be recovered from the data. Our identification of WG protein in adjacent cells as a regulator and our failure to detect activation or repression by *engrailed* (whose transcription depends on WG) suggest that WG may have a more direct effect on the regulation of *ptc*. Further, it is possible that existence of regulation by *en* is inconclusive, given the data, especially in light of the discrepancy between the Boolean and continuous models. Currently we have no evidence to support regulation by PTC or by PH.

For the protein complex PH, we detected that its synthesis is regulated itself

and by *ptc*, *hh* and their corresponding products in adjacent cells. Recall that PH is the molecule formed when HH from adjacent cells binds to the transmembrane receptor PTC. In (Albert and Othmer, 2003) the authors assume in their model that this binding occurs instantaneously since it is known that the reaction occurs faster than transcription or translation (which they also presuppose to require 1 time unit for completion). Therefore, we infer that the synthesis of PH at a given time step is determined by the expression levels of *ptc*, *hh* and *hh* product, from neighboring cells, in the previous time step. However, we did not identify PTC as a direct regulator. As PH may be transported into other cells via exocytosis (Taylor *et al.*, 1993), its expression levels may not be maintained. This suggests that its presence in a cell signals its transport out of the cell, hence implying a type of "autoregulation".

We identified the gene *ptc*, its product, the complex PH, and extracellular *hh* to affect synthesis of SMO. The binding of PTC to SMO inactivates it via post-translational modification (Ingham, 1998). However, if PTC binds to HH, then SMO is activated (Ingham, 1998). So the translation of *smoothened* mRNA is dependent on the synthesis of the complex PH. (The mRNA *smoothened* was not included in the model since the gene is transcribed ubiquitously throughout the segment (Ingham, 1998) and its transcription is not regulated by any biochemical in the model.) Since this binding is assumed to occur instantaneously, as described above, we instead detect the regulation by *hh* in adjacent cells and not its product.

While we correctly identified regulation by SMO, CI, and *hh* in neighboring cells for the activating (CIA) and repressing (CIR) transcriptional factors transformed from CI, we predicted two other sources of regulation, namely by PTC and PH. There is evidence that PH activates signalling by SMO (Chen and Struhl, 1998), causing the transformation of CI into CIA, inhibiting cleavage of CI. Further, the rate of cleavage is a function of the amount of free PTC in the cell (von Dassow *et al.*, 2000). The presence of the links from PTC and PH to CIA and CIR suggests that their indirect regulation can be detected in the data.

To summarize the performance of our algorithm for detecting network topology, we show that the use of expression levels of wildtype and null mutants enables detection of about 84% of the interactions in the Boolean model. In contrast, using data only from the wildtype yields a mere 32% identification.

Next we focus on reverse engineering the dynamics of the Boolean model, that is, the Boolean functions on each of the network nodes. As pointed out above, the functions in the Boolean model contain terms that evaluate to 0 on all input data, so that we are unable to detect the corresponding relationships. In order to compare the dynamics predicted by our method with the Boolean model we reduce the polynomials in Table B.2 by removing vanishing terms, as

described in the previous section. Note, however, that this reduction depends on the choice of term ordering, as does the output of our algorithm. For each choice of term ordering the reduced functions of the Boolean model and the functions reverse engineered from the data agree exactly (see Table 3). This shows that we are able to completely predict the Boolean model's dynamics from the wildtype and knock-out data. However, due to the sensitivity of our method to the chosen term ordering, the particular form of the reverse-engineered functions may not be directly interpretable in terms of regulatory relationships. We therefore proceed as before by extracting information about nonlinear terms containing products of more than one variable common to the reverse-engineered functions for multiple term orderings. These are analogous to mixed terms in ODE models.

In the consensus model constructed from the wildtype, all polynomials are linear, suggesting that the concentrations of all molecular species in the network are regulated by biochemicals working independently. However, Albert *et al.* (2003), von Dassow *et al.* (2000), and others argue differently, as indicated in their models. For a more accurate dynamics, we include expression data for the null mutants.

In the consensus model built with knock-out data, we found the following nonlinear interactions. For the proteins SMO and PH, we found that their synthesis depends on the "interaction" between the genes *ptc* and extracellular *hh*. Specifically, the terms $[ptc] * [hh_{i-1}]$ and $[ptc] * [hh_{i+1}]$ appear in the polynomial functions for SMO and PH, for all term orders, which also appear in the Boolean functions for these two proteins. In the function describing transcription of *wg*, the terms $[wg]*$[CIA] and $[wg]*$[CIR] are present, implying that its expression results from interactions between itself and the activator and repressor forms of the protein CI. These products also appear in the Boolean function for *wg*. Furthermore, the function of the protein fragments of CI include the terms [PTC]$*[hh_{i-1}]$ and [PTC]$*[hh_{i+1}]$, both of which exist in the corresponding Boolean functions. Finally, in any cell, our method detects that PTC and *hh* from a neighboring cell interact, affecting the transcription of *ptc*. As mentioned above, we have no evidence to support this implication.

Our method shows a marked improvement when knock-out data are included. We were able to reconstruct about 84% of the topology of the Boolean network, versus only 32% when knock-out data are not included. Further, we identified 10 nonlinear interactions, versus none in the model constructed with only wildtype data.

$$
\begin{aligned}
f_6 &= x_5 \\
f_7 &= x_6 \\
f_8 &= x_{12}x_{13} + x_{13}x_{16} + x_{13}x_{20} + x_{18}x_{20} + x_{13}x_{21} + x_{19}x_{21} + x_{13} + x_{18} + x_{19} \\
f_9 &= x_{10}x_{14} + x_{14}x_{18} + x_{14}x_{19} + x_9x_{20} + x_{18}x_{20} + x_9x_{21} + x_{19}x_{21} + x_8 + x_9 + x_{10}
\end{aligned}
$$

Table 3
Sample Boolean functions reduced by the ideal of wildtype and knock-out time series.

4 Data Discretization

Since gene expression data are real numbers, the first step in any reverse-engineering algorithm using discrete models must be to discretize these real numbers into a finite (typically small) set of possible states. For Boolean networks this simply amounts to the choice of a single threshold for the expression level of each gene, below which a gene is considered inactive. The issue of data discretization has been studied extensively, in particular from the points of view of Bayesian network applications and machine learning; see, e.g., (Dougherty *et al.*, 1995) and (Friedman and Goldszmidt, 1996). Obviously, the way one discretizes the data plays an important role in what model one obtains. The first important choice is the number of discrete states allowed. In our case, this is the choice of p in the algorithm. It follows from results in (Green, 2003) that for p large enough the result of the reverse-engineering algorithm does not depend on p anymore, in the sense that the monomials in the polynomials remain the same, possibly with different coefficients. While that paper does not give an algorithm for choosing a suitable p, extensive experiments with networks simulated with the biochemical network simulation program *Gepasi* (`http://www.gepasi.org`) suggest that for data sets up to 50 nodes an optimal p is in the range between 5 and 13. The effect of the choice of p is demonstrated in Figures 3 and 4. This example also demonstrates that the availability of more than two discrete states can be very helpful for modeling purposes.

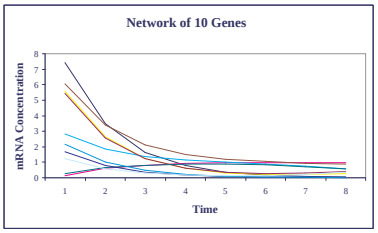

Fig. 3. The graph of the real-valued time series for a network $\mathcal{G}$ of 10 genes.

There are various ways to attach a discrete label to real-valued data. Thresholds with biological relevance is one type of labeling that can be used. For example, up-regulation, no regulation, and down-regulation of a gene are

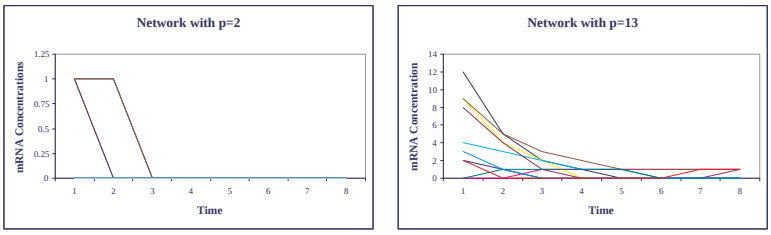

Fig. 4. The graph of the discrete time series for $\mathcal{G}$: $p = 2$ (left) and $p = 13$ (right) with discretization method = global as described above.

two thresholds that can partition a given data set into 3 groups, with labels $1, -1, 0$, respectively. This maps the real-valued data into values in $\mathbb{Z}/3$. More thresholds can be integrated to refine the partitioning of the data set. Another method of discretization is to normalize the expression of each gene or protein and use the deviation from the mean to discretize the data.

5 Effect of Noise on the Algorithm

In the previous section, one can see that the choice of prime provides different levels of resolution of the real-valued time series, yielding sharp differences in the discrete time series. To minimize this effect, a suitably large prime should be considered (Green, 2003). However, data collected from experimental processes typically have errors introduced due to deficiencies in methodology or imperfections in instrumentation, as is the case in microarray data. Furthermore, biological systems, such as biochemical pathways, exhibit variability in population or concentration levels. For our purposes it is important to quantify this noise.

Because our method begins by discretizing real-valued time series, one would expect that discretization will smooth out some of the noise. To test this hypothesis, we incorporated random noise, using a normal distribution, into time series generated from 100 simulated networks generated by the AGN software described in (Mendes *et al.*, 2003). We generated 25 networks with 20 nodes for each of the 4 topologies supported by AGN. All time series are of length 11 time steps. We added noise in two ways: first, as a percentage of each data point, simulating biological variation, and second, as a percentage of a fixed value, simulating instrumentation error. In both cases, we added 10% and 25% noise. We chose 3 different thresholds (0.5, 1.0, and 2.5) with which to Booleanize all time series. Then we computed the median number of entries that were changed in the noisy time series for both 10% and 25% noise, as compared to the original time series.

The median number of entries that changed for all experiments is 5 (2.3% of

220). In the case when noise is a percentage of each data point, the median is 2 (1% of 220), when the threshold was restricted to 0.5 or 2.5. For the experiments described above, we infer that 10% of the noise added is propogated to the discrete time series.

To test for the effect of noise on our method, we added 1% noise to the discrete time series for the Boolean functions in (Albert and Othmer, 2003). (We estimate this to correspond to approximately 10% noise in the "real" data, given the above results.) We then applied our method to the noisy time series and used 8 variable orderings to construct a consensus model. For 6 of the nodes (SLP, *wg*, WG, *en*, HH, and CI) we were able to identify all of the correct links, plus some false positives. For 5 of the nodes (*ptc*, PTC, SMO, CIR, and CIA), we identified half of the correct links. For the remaining nodes, we had no conclusive results.

6 Discussion

We have described a new method to discover regulatory relationships between the nodes in a gene regulatory network from data. The modeling framework is that of time-discrete dynamical systems with finite (but possibly large) state sets for the variables. The crucial further assumption is that the set of possible states for a variable can be given the structure of a finite field. As a consequence, we have available to us the very well-developed theory of algorithmic polynomial algebra, with a variety of implemented procedures. It is this machinery that we employ for the task of reverse engineering. In giving up a bit of freedom in the choice of state sets, we gain a mathematical framework well suited for reverse-engineering purposes. Our method is very similar to that in (Yeung *et al.*, 2002), in that we first compute all possible networks that fit the given data. We do this not by enumeration, but rather by a procedure similar to describing all solutions of a homogeneous system of linear equations by computing a basis for the nullspace. Then, as in (Yeung *et al.*, 2002), we choose a particular network. Our selection criterion is to choose polynomial functions that do not contain any terms which are identically equal to 0 on the given data set. So, we choose a minimal network, not in terms of network connections like in (Yeung *et al.*, 2002) and (Liang *et al.*, 1998), but rather in terms of the structure of the functions. The rationale is that if a term vanishes on the data input, then no reverse-engineering method should be able to identify it without prior biological knowledge.

We contrast our method with the discrete methods described in the introduction. In its essence, the Thieffry-Thomas model can also be represented as a function from a finite set of states to itself, with dynamics generated by iteration of this function. The lack of any further mathematical structure in the

model makes its analysis very difficult. Moreover, this modeling framework does not lend itself to mathematical reverse-engineering methods. Even in the much simpler case of Boolean networks, all the algorithms discussed above rely at some point on enumeration of a large number of possible components of the putative network.

For method validation we used the Boolean model of the *D. melanogaster* segment polarity network recently published in (Albert and Othmer, 2003). We generated time series and perturbed time series from this model and used them as input for the algorithm. Note that for our purposes it is irrelevant whether the model is in fact an accurate depiction of the real system. Of the 44 links in the model we correctly identify 37, together with 13 links not present in the model. For each of the links that our method identifies incorrectly, we provide evidence as to possible reasons. Furthermore, we are able to identify many of the key nonlinear relations among variables in the model.

In the absence of good approximation methods over finite fields, our method does exact fitting of data and is consequently quite sensitive to noise. Future work includes the development of approximation methods that will alleviate some of this sensitivity. A more systematic study of different data discretization methods and their effect on noise reduction will also be carried out. Finally, the method will be validated using both simulated networks and published models that generate real-valued data.

7 Acknowledgements

The research in this paper was partly supported by NSF grants DMS-0138323 and DMS0083595, and NIH grant RO1GM068947-01. The authors thank J. Shah for the implementation of the algorithms discussed in this paper, and O. Colón-Reyes, A. de la Fuente, L. Garcia, E. Green, A. Li, P. Mendes, E. Nordberg, J. Snoep, M. Stillman, and B. Sturmfels for helpful discussions. Finally, the authors thank the anonymous referees for their helpful comments.

References

[1] Akutsu, T., Miyano, S., and Kuhara, S., 1999. Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. In Altman, R. B., Lauderdale, K., Dunker, A. K., Hunter, L., and Klein, T. E. (eds.), *Proc. Pacific Symp. Biocomput.* **4**, 17–28, World Scientific, Singapore.

[2] Albert, R. Othmer, H., 2003. The topology of the regulatory interac-

tions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *J. Theor. Biol.* **223**, 1–18.

[3] Bower, J. M., and Bolouri, H., 2001. *Computational Modeling of Genetic and Biochemical Networks*, The MIT Press, Cambridge, MA.

[4] Brazma, A. and Schlitt, T., 2003. Reverse engineering of gene regulatory networks: a finite state linear model. preprint, available at `http://genomebiology.com/2003/4/6/P5`.

[5] Cadigan, K.M., Grossniklaus, U., Gehring, W.J., 1994. Localized expression of *sloppy paired* protein maintains the polarity of *Drosophila* parasegments. *Genes Dev.* **8**, 899–913.

[6] Chen, Y., and Struhl, G., 1998. In vivo evidence that Patched and Smoothened constitute distinct binding and transducing components of a Hedgehog receptor complex. *Development* **125**, 4943–4948.

[7] Cho, R., Campbell, M., Winzeler, E., Steinmetz, L., Conway, A., Wodicka, L., Wolfsberg, T., Gabrielan, A., Landsman, D., Lockhart, D., and Davis, R., 1998. A genome-wide transcriptional analysis of the mitotic cell cycle. *Mol. Cell* **2**, 65–73.

[8] Cox, D., Little, J., and O'Shea, D., 1997. *Ideals, Varieties, and Algorithms*, Springer Verlag, New York.

[9] de la Fuente, A., Brazhnik, P., and Mendes, P., 2002. Linking the genes: inferring quantitative gene networks from microarray data. *Trends in Genetics* **18**, 395–398.

[10] de Jong, H., 2002. Modeling and simulation of genetic regulatory systems: a literature review. *J. Comp. Biol.* **9**, 67–103.

[11] Dougherty, J., Kohavi, R., and Sahami, M., 1995. Supervised and unsupervised discretization of continuous features. In Prieditis, A. and Russell, S. (eds.), *Machine learning: Proceedings of the 12th International Conference*, Morgan Kauffman, San Francisco, CA.

[12] Filkov, V., Skiena, S., and Zhi, J., 2002. Analysis techniques for microarray time-series data. *J. Comp. Biol.* **9**, 317–330.

[13] Filkov V., and Istrail, S., 2002. Inferring gene transcription networks: the Davidson model. *Genome Informatics* **13**, 236–239.

[14] Friedman N., Linia, l. N. M., Nachman, I., and Pe'er D., 2000. Using Bayesian networks to analyze expression data, *J. Comp. Biol.* **7**, 601–620.

[15] Friedman, N., and Goldszmidt, M., 1996. Discretization of continuous attributes while learning Bayesian networks. In Saitta, L. (ed.), *Proc. of the 13th International Conference on Machine Learning*, 157–165, Morgan Kauffman, San Francisco, CA.

[16] Green, E. 2003. On polynomial solutions to reverse-engineering problems. Preprint.

[17] Hartemink, A. J., Gifford, D. K., Jaakkola S., and Young, R. A., 2001. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. *Pac. Symp. Biocomput.*, World Scientific, Singapore.

[18] Hooper, J.E., Scott, M.P., 1992. The molecular genetic basis of positional information in insect segments, in: Hennig, W. (Ed.), *Early Embryonic Development of Animals*, Springer, Berlin, 1-49.

[19] Ingham, P. W., 1998. Transducing hedgehog: the story so far. *EMBO J.* **17**, 3505-3511.

[20] Ingham, P. W., McMahon, A. P., 2001. Hedgehog signaling in animal development: paradigms and principles. *Genes Dev. 15*, 30593087.

[21] Kauffman, S. A., 1969. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.* **22**, 437–467.

[22] Kitano, H., 2002. Systems biology: A brief overview. *Science* **295**, 1662–1664.

[23] Lang, S., 1971. *Algebra*. Addison Wesley, Reading, MA.

[24] Lewis, J.E., and Glass, L., 1991. Steady states, limit cycles, and chaos in models of complex biological networks. *Int. J. Bifurcation and Chaos* **1**, 477–483.

[25] Liang, S., Fuhrman, S., and Somogyi, R., 1998. REVEAL, a general reverse engineering algorithm for inference of genetic network architectures. *Pac. Symp. Biocomput.* **3**, 18–29. World Scientific, Singapore.

[26] Lidl, R., and Niederreiter, H., 1997. *Finite Fields*, Encyclopedia of Mathematics and its Applications 20, 2nd Edition, Cambridge University Press.

[27] Marchand, H., and Le Borgne, M., 1998. Partial order control of discrete event systems modeled as polynomial dynamical systems. in *IEEE International Conference on Control Applications*, Trieste, Italy.

[28] Mendes, P., Sha, W., and Ye, K., 2003. Artificial gene networks for objective comparison of analysis algorithms. *Bioinformatics* **19**, 122–129.

[29] Mendoza, L., Thieffry, D., and Alvarez-Buylla, E. R., 1999. Genetic control of flower morphogenesis in *Arabidopsis thaliana*: a logical analysis. *Bioinformatics* **15**, 593–606.

[30] Repsilber, D., Liljenström, H., and Andersson, S. G. E., 2002. Reverse engineering of regulatory networks: simulation studies on a genetic algorithm approach for ranking hypotheses. *BioSystems* **66**, 31–41.

[31] Robbiano L., 1998. Gröbner bases and statistics, in *Gröbner Bases and Applications*, Buchberger B. and Winkler F. (eds.), Cambridge University Press, New York.

[32] Schmulevich, I., Dougherty, E. R., Kim, S., and Zhang, W., 2002. Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics* **18**, 261–274.

[33] Shparlinski, I. E., 1999. *Finite Fields: Theory and Computation*. Kluwer Academic Publishers, Dortrecht.

[34] Spellman, P. T., Sherlock, G., Zhang, M. Q., Iyer, V. R., Anders, K., Eisen, M. B., Brown, P. O., Botstein, D., and Futcher, B., 1998. Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Mol. Biol. Cell* **9**, 3273–3297.

[35] Taylor, A.M., Nakano, Y., Mohler, J., Ingham, P.W., 1993. Contrasting

distributions of patched and hedgehog proteins in the *Drosophila* embryo. *Mech. Dev.* **42**, 89–96.

[36] Thieffry, D., and Thomas, R., 1998. Qualitative analysis of gene networks. *Proc. Pacific Symp. on Biocomputing*, 77–88, World Scientific, Singapore.

[37] Thieffry, D., Thomas, R., and Kaufman, M., 1995. Dynamical behaviour of biological regulatory networks–I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state. *Bulletin of Math. Bio.* **57**, 247–276.

[38] Thomas, R., 1991. Regulatory networks seen as asynchronous automata: a logical description. *J. Theor. Biol.* **153**, 1–23.

[39] von Dassow, G., Meir, E., Munro, E. M., Odell, G. M., 2000. The segment polarity network is a robust developmental module. *Nature* **406**, 188–192.

[40] Yeung. M. K. S., Tegnér, J., and Collins, J. J., 2002. Reverse engineering gene networks using singular value decomposition and robust regression. *Proc. Natl. Acad. Sci.* **99**, 6163–6168.

Appendix

A Mathematical Background

In this section we give the mathematical details of the reverse-engineering algorithm and some basic facts about computational algebra relevant to this paper. The basic problem that lies at the heart of computational algebra is that long division of multivariate polynomials differs from that of univariate polynomials in that the remainder is not uniquely determined. It depends on several choices that need to be made along the way. Univariate division of a polynomial f by another one, g, proceeds by dividing the highest power of the variable in g into the highest power in f. For multivariate polynomials there are many choices of ordering the terms of a polynomial, which affects the outcome of the division. Also, when dividing more than one polynomial into a given one, the outcome typically depends on the order in which the division is carried out. To be precise, let k be a field, e.g. the field of real or complex numbers, or a finite field, and let $f, f_1, \ldots, f_m \in k[x_1, \ldots, x_n]$ be polynomials in the variables $x_1, \ldots, x_n$. The question whether there are polynomials $g_1, \ldots, g_m \in k[x_1, \ldots, x_n]$ such that

$$f = \sum_{i=1}^{m} g_i f_i$$

can in general not be answered algorithmically. In the language of abstract algebra, let $I = \langle f_1, \ldots, f_m \rangle$ be the *ideal* in $k[x_1, \ldots, x_n]$ generated by the f_i, that is, I is the set of all linear combinations $\sum g_i f_i$, with $g_i \in k[x_1, \ldots, x_n]$. Then we are asking whether f is an element of I. This is known as the *ideal membership problem*.

In general, the ideal I can be generated by sets of polynomials other than the f_i, similar to a vector space possessing many different bases, and the solution to the ideal membership problem does not depend on the choice of a particular generating set. It turns out that if one chooses a very special type of generating set, known as a *Gröbner basis*, for the ideal, then the ideal membership problem becomes solvable algorithmically. There is a basic algorithm to compute a Gröbner basis for an ideal, the Buchberger algorithm. Using this algorithm as a foundation, many problems in multivariate computational algebra can be solved algorithmically, including the computation of intersections of ideals, which is the key computation we are using in our algorithm. The central ingredient in this algorithm is the Elimination Theorem (see (Cox *et al.*, 1997), p. 113). This theorem is also the source of the very high computational complexity of many algorithms, since it requires the use of a computationally expensive type of term ordering. For details see, e.g., (Cox *et al.*, 1997).

As described earlier, the basic idea of our algorithm is as follows. First we choose a term order for $k[x_1, \ldots, x_n]$. This is necessary for all subsequent calculations. To compute the ideal I of all polynomial functions that are identically equal to 0 on a given collection $\{\mathbf{s_i}\}$ of points we proceed as follows. Let I_i be the ideal of all polynomials that take on the value 0 on $\mathbf{s}_i$. It is straightforward to see that I_i contains the ideal

$$\langle x_1 - s_{i1}, \ldots, x_1 - s_{in} \rangle.$$

But this ideal is maximal with respect to inclusion, so that it has to be equal to I_i. Then the ideal I is equal to the intersection

$$I = \bigcap_{i=1}^{m} I_i.$$

Algorithms to compute the intersection of ideals are implemented in many specialized computer algebra systems. As mentioned earlier, we have used *Macaulay2* for all computations. Finally the reduction of the special solution f_0 modulo I is simply the remainder of f_0 under division by a reduced Gröbner basis of I. This too is a standard algorithm implemented in most computer algebra systems. It is important to observe that, if g_0 is another particular solution to the interpolation problem, then f_0 and g_0 differ by a polynomial in I, as shown before. Therefore, the reduction of f_0 by I is equal to the reduction of g_0. Consequently, it does not matter which method we use to construct a particular solution.

B Tables

$$
\begin{array}{lcl}
f_1 & = & SLP_i^{t+1} = \begin{cases} 0 \ if \ i \bmod 4 = 1 \ or \ i \bmod 4 = 2 \\ 1 \ if \ i \bmod 4 = 3 \ or \ i \bmod 4 = 0 \end{cases} \\
f_2 & = & wg_i^{t+1} = \left(CIA_i^t \wedge SLP_i^t \wedge \neg CIR_i^t\right) \vee \left(wg_i^t \wedge \left(CIA_i^t \vee SLP_i^t\right) \wedge \neg CIR_i^t\right) \\
f_3 & = & WG_i^{t+1} = wg_i^t \\
f_4 & = & en_i^{t+1} = \left(WG_{i-1}^t \vee WG_{i+1}^t\right) \wedge \neg SLP_i^t \\
f_5 & = & EN_i^{t+1} = en_i^t \\
f_6 & = & hh_i^{t+1} = EN_i^t \wedge \neg CIR_i^t \\
f_7 & = & HH_i^{t+1} = hh_i^t \\
f_8 & = & ptc_i^{t+1} = CIA_i^{t+1} \wedge \neg EN_i^{t+1} \wedge \neg CIR_i^{t+1} \\
f_9 & = & PTC_i^{t+1} = ptc_i^t \vee \left(PTC_i^t \wedge \neg HH_{i-1}^t \wedge \neg HH_{i+1}^t\right) \\
f_{10} & = & PH_i^{t+1} = PTC_i^{t+1} \wedge \left(HH_{i-1}^{t+1} \vee HH_{i+1}^{t+1}\right) \\
f_{11} & = & SMO_i^{t+1} = \neg PTC_i^{t+1} \vee HH_{i-1}^{t+1} \vee HH_{i+1}^{t+1} \\
f_{12} & = & ci_i^{t+1} = \neg EN_i^t \\
f_{13} & = & CI_i^{t+1} = ci_i^t \\
f_{14} & = & CIA_i^{t+1} = CI_i^t \wedge \left(SMO_i^t \vee hh_{i-1}^t \vee hh_{i+1}^t\right) \\
f_{15} & = & CIR_i^{t+1} = CI_i^t \wedge \neg SMO_i^t \wedge \neg hh_{i-1}^t \wedge \neg hh_{i+1}^t
\end{array}
$$

Table B.1
Boolean functions for the network in Figure 1.

$$
\begin{aligned}
f_1 &= x_1 \\
f_2 &= x_1x_{14}\left(x_{15}+1\right)+x_2\left(x_1+x_{14}+x_1x_{14}\right)\left(x_{15}+1\right)+x_1x_2x_{14}\left(x_{15}+1\right)^2\left(x_1+x_{14}+x_1x_{14}\right) \\
f_3 &= x_2 \\
f_4 &= \left(x_{16}+x_{17}+x_{16}x_{17}\right)\left(x_1+1\right) \\
f_5 &= x_4 \\
f_6 &= x_5\left(x_{15}+1\right) \\
f_7 &= x_6 \\
f_8 &= x_{13}\left(\left(x_{11}+x_{20}+x_{11}x_{20}\right)+x_{21}+\left(x_{11}+x_{20}+x_{11}x_{20}\right)x_{21}\right)\left(x_4+1\right) \\
&\quad \left(x_{13}\left(x_{11}+1\right)\left(x_{20}+1\right)\left(x_{21}+1\right)+1\right) \\
f_9 &= x_8+x_9\left(x_{18}+1\right)\left(x_{19}+1\right)+x_8x_9\left(x_{18}+1\right)\left(x_{19}+1\right) \\
f_{10} &= \left(x_8+x_9\left(x_{18}+1\right)\left(x_{19}+1\right)+x_8x_9\left(x_{18}+1\right)\left(x_{19}+1\right)\right)\left(x_{20}+x_{21}+x_{20}x_{21}\right) \\
f_{11} &= x_8+x_9Y+x_8x_9Y+1+x_{20}+\left(\left(x_8+x_9Y+x_8x_9Y+1\right)x_{20}\right)+x_{21} \\
&\quad +\left(x_8+x_9Y+x_8x_9Y+1+x_{20}+\left(x_8+x_9Y+x_8x_9Y+1\right)x_{20}\right)x_{21} \\
f_{12} &= x_5+1 \\
f_{13} &= x_{12} \\
f_{14} &= x_{13}\left(\left(x_{11}+x_{20}+x_{11}x_{20}\right)+x_{21}+\left(x_{11}+x_{20}+x_{11}x_{20}\right)x_{21}\right) \\
f_{15} &= x_{13}\left(x_{11}+1\right)\left(x_{20}+1\right)\left(x_{21}+1\right)
\end{aligned}
$$

SLP	wg	WG	en	EN	hh	HH	ptc	PTC	PH	SMO
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
ci	CI	CIA	CIR	WG_{i-1}	WG_{i+1}	HH_{i-1}	HH_{i+1}	hh_{i-1}	hh_{i+1}	Y
x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}	$\left(x_{18}+1\right)\left(x_{19}+1\right)$

Table B.2
Polynomial representations of the Boolean functions in Table B.1, together with the legend of variable names.

$$
\begin{aligned}
f_1 &= x_1 \\
f_2 &= x_1x_{14} + x_2x_{14} + x_2x_{15} + x_2 \\
f_3 &= x_2 \\
f_4 &= x_{16} \\
f_5 &= x_4 \\
f_6 &= x_5 \\
f_7 &= x_6 \\
f_8 &= x_{12}x_{13} + x_{13}x_{16} + x_{13}x_{20} + x_{18}x_{20} + x_{13}x_{21} + x_{19}x_{21} + x_{13} + x_{18} + x_{19} \\
f_9 &= x_{10}x_{14} + x_{14}x_{18} + x_{14}x_{19} + x_9x_{20} + x_{18}x_{20} + x_9x_{21} \\
&\quad + x_{19}x_{21} + x_8 + x_9 + x_{10} \\
f_{10} &= x_{10}x_{14} + x_{14}x_{18} + x_{14}x_{19} + x_8x_{20} + +x_8x_{21} + x_{10} + x_{18} + x_{19} \\
f_{11} &= x_8x_{20} + x_9x_{20} + x_{18}x_{20} + x_8x_{21} + x_9x_{21} + x_{19}x_{21} + x_8 + x_9 + x_{18} + x_{19} + 1 \\
f_{12} &= x_5 + 1 \\
f_{13} &= x_{12} \\
f_{14} &= x_{11}x_{13} + x_9x_{20} + x_{18}x_{20} + x_9x_{21} + x_{19}x_{21} + x_{10} + x_{18} + x_{19} \\
f_{15} &= x_{11}x_{13} + x_9x_{20} + x_{18}x_{20} + x_9x_{21} + x_{19}x_{21} + x_{10} + x_{13} + x_{18} + x_{19}
\end{aligned}
$$

Table B.3
Boolean functions reduced by the ideal of wildtype and knock-out time series.